Surface Motion Capture Animation Synthesis

Adnane Boukhayma and Edmond Boyer

Abstract—We propose to generate novel animations from a set of elementary examples of video-based surface motion capture, under user-specified constraints. 4D surface capture animation is motivated by the increasing demand from media production for highly realistic 3D content. To this aim, data driven strategies that consider video-based information can produce animation with real shapes, kinematics and appearances. Our animations rely on the combination and the interpolation of textured 3D mesh data, which requires examining two aspects: (1) Shape geometry and (2) appearance. First, we propose an animation synthesis structure for the shape geometry, the Essential graph, that outperforms standard Motion graphs in optimality with respect to quantitative criteria, and we extend optimized interpolated transition algorithms to mesh data. Second, we propose a compact view-independent representation for the shape appearance. This representation encodes subject appearance changes due to viewpoint and illumination, and due to inaccuracies in geometric modelling independently. Besides providing compact representations, such decompositions allow for additional applications such as interpolation for animation.

Index Terms—Character animation, 3D video, multiview reconstruction, video-based animation, 4D modeling, 4D performance capture

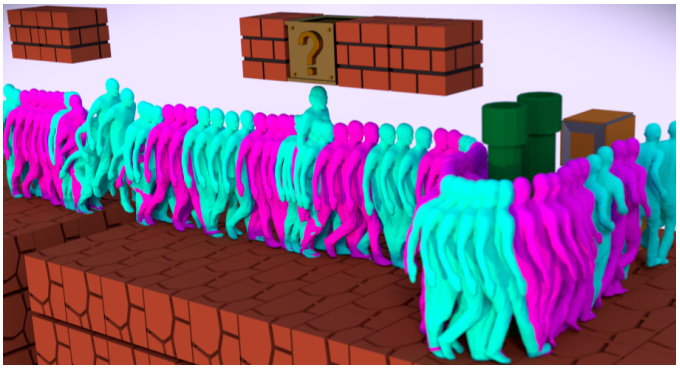


Fig. 1: 4D Animation satisfying environment constraints. Red frames are synthetic and green frames are original.

1 Introduction

After more than a decade of predominance of standard motion capture, we have witnessed in the recent years the emergence of 4D(3D+t) surface capture either through high quality multi-view set-ups [1], [2], [3], [4], [5] or in a more democratized form with low-cost sensors [6], [7], [8]. Such data provides real full shape, appearance and kinematic information of the dynamic object of interest and hence can find many uses for 3D content generation, augmentation of machine learning training sets or statistical modeling, among other examples. In this work, we propose a solution for building animations using this type of data. As illustrated in Figure 3, our input is a set of temporally consistent mesh sequences with their texture maps, also referred to as 4D models, and the output is novel data of the same nature satisfying high-level constraints. Figure 2 illustrates the steps involved in creating our input 4D textured models. Frame-dependant surface reconstruction [9] is performed using videos from different viewpoints. A 2D mapping [10] of a template geometry is used to obtain a texture atlas. The reconstructed sequences are tracked [11] with the same template to obtain temporally consistent meshes. The temporally tracked sequence geometries are then used to back-project the appearance information from the input views and into atlas-consistent texture maps [12] thus allowing us to render textured 4D sequences.

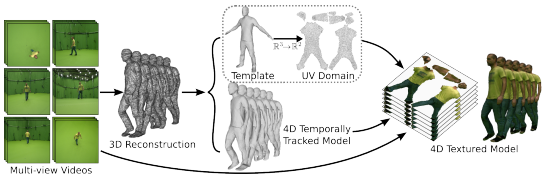


Fig. 2: Data acquisition pipeline.

The main difficulties involved in textured shape animation are first finding an optimal way of blending and concatenating shape geometry instances to obtain the desired animations, and second storing shape appearance instances compactly and interpolating them without ghosting artifacts. Regarding shape animation, we limit the synthesis to simple data reuse which consists of concatenating original motion segments and synthetic transitions between them. Hence we need a way to synthesize visually plausible transition segments (Section 3), and we need a structure to control the animation process with respect to user-guidance by identifying good transitions (Section 4). For transitions, we extend some of the techniques previously proposed for skeletal motion capture data and adapt them to mesh data using differential mesh processing [13] to account for shape non-rigidities. For transition selection and control, we consider animations that are globally optimal with respect

• *The authors are with Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble France.*
E-mail: adnane.boukhayma@gmail.com.

to a defined realism criterion, and propose a structure called the Essential graph that proves to be more optimal than standard Motion graphs. The realism criterion accounts for a cost that tries to preserve shape consistency along a motion segment, while minimizing the number of poses within a segment. We focused on the optimality of the animation structure as our data is more intricate and sensitive to editing compared to skeletal motion capture.

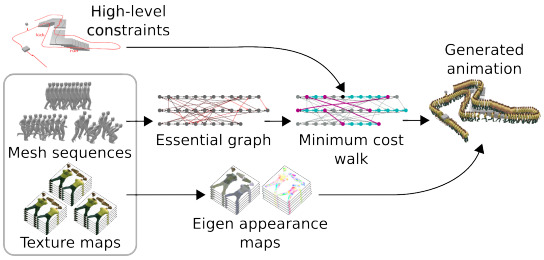


Fig. 3: Animation pipeline.

Regarding appearance animation, we propose a view-independent appearance representation and estimation algorithm, to encode the appearance variability of the dynamic subject. Compactly representing image data from all frames and viewpoints of the subject can be seen as a non-linear dimensionality reduction problem in image space, where the main non-linearities are due to the underlying scene geometry. By coarsely registering our dynamic shape with a template, for which we pre-computed a single reference surface-to-texture unwrapping (Figure 2), we reduce the problem to a single texture space by coping with remaining fine-scale misalignments due to registration errors using realignment warps in the texture domain. Because they encode residual geometric variations, these warps are decomposed using linear PCA, yielding Eigen warps. After alignment, we assume that the remaining image variabilities can be straightforwardly identified with linear PCA and thus we obtain Eigen textures. The full appearance information of all subject sequences can then be compactly stored as linear combinations of Eigen textures and Eigen warps. This method is shown to yield effective estimation performance. In addition, the learned texture and warp manifolds allow for efficient generalizations, such as texture interpolations to generate new unobserved content from blended input instances, which are appended to interpolated poses obtained from the shape animation pipeline (Figure 3).

This paper builds on our previous works [14] [15] and combines them to obtain full shape and appearance 4D animation. We additionally extend these works by further evaluation, discussion and elaboration of technical details. Regarding appearance, we notably extend the numerical evaluation of compactness (Figure 18) and generalizability (Figure 20) of the representation in [15] in section 5.2, and the impact of texture reduction on interpolation quality (Figure 22) in section 5.3. In the shape animation part of the pipeline [14], we provide numerical evaluations of the shape pose distance in Section 3.1 (Table 1), transition optimization method in Section 3.3 (Figure 7) and the organizing data structure in Section 4.1 (Figure 9). We additionally propose algorithms for transition generation (Algorithm 1) and animation synthesis (Algorithm 2) in Section 4.2.

2 Related work

Previous works proposed similar pipelines for animation synthesis (e.g. [16], [17]). This paper contributes to the state-of-the-art on the following aspects: we propose a globally optimal data structure (Section 4.1) that differs from the traditional motion graphs [17]; we introduce new motion transitions that combine dynamic time warping and varying blended segments lengths (Section 3.3); we improve extrinsic alignment for motion blending compared to existing methods [17] (Section 4.2); we present a character appearance model that mutualises appearance information from several captured sequences (Section 5).

Motion data structure

Traditional motion capture systems provide sparse information on moving shapes. A large body of work is considering such input data to animate virtual characters. To this aim, Motion graphs [18], [19], [20] are an ubiquitous tool for organizing recorded movement sequences of a shape, and many improvements to this structure have been proposed such as *Interpolated* [21], *Well-connected* [22] or *Optimization based* [23] Motion graphs. A Motion graph is composed of nodes for shape poses and edges that represent transitions between them. New transitions are added by selecting local minima in transition cost matrices between sequence pairs. Several works extend Motion graphs to mesh data. [24] represents human shapes as geometry images using spherical parametrizations. [25] builds Motion graphs for meshes without interpolations. Our previous work [14] only solves for geometry animation. [17] proposes animation for unstructured textured meshes and their appearances. The work of [16] extends parametric Motion graphs [26] to mesh data and interpolates appearances as well as shapes, and [27] extends this framework to Mocap driven character animation. We note that parametric Motion graphs solve a different problem than Motion graphs, as they are more supervised in the sense that they require building parametric motion spaces with logically compatible motions, prior to connecting them with each other. On the contrary, Motion graphs are unsupervised, as their construction requires no prior knowledge on the nature of the motion sequences involved. In this work, we also solve the unsupervised animation scenario for geometry and appearance animation using a graph representation. However, we use a different graph structure [14] that improves over standard Motion graphs in global optimality with respect to surface deformation and transition duration.

Motion transition

Motion transition generation is an important component of motion synthesis. Although recent works [28], [29] leverage the abundance of skeletal Mocap data to learn such tasks, gradual frame blending remains yet a reasonable and intuitive solution that is widely used in the literature, either in a *Start-end* [18], *Center-aligned* [20] or *Left aligned* [19], [21] form. This method is however prone to artifacts. These limitations can be substantially reduced with dynamic time warping prior to interpolation. Following a strategy similar to [30], we extend dynamic time warping to transitions between mesh sequences. In addition, inspired by the study of [31] that emphasizes the importance of suitable transition

durations, we also optimize such durations within transitions.

Appearance animation

Appearance of 3D shapes can be stored in two main forms: View-dependent texturing resamples from each initial video frame [32] with additional alignments to avoid ghosting effects [33], but this strategy appears to be memory costly as it requires storing all images from all viewpoints. Instead, texture maps can be built offline from all viewpoints [1] thus reducing storage, with 3D superresolution techniques [12], [34], [35] to overcome resolution and sampling limitations. For animation synthesis, using only one template texture map results in substantial realism loss. Using topologically time-coherent geometries with consistent 2D unwarpings allow appearance information fusion over time and a more compact storage than per-frame textures. Some works consider this aspect but only for short segments of adjacent frames. [12] attempts to enhance texture resolution, [36] uses a view-based multi-layer texture map representation, and [1] uses tracked surfaces to improve compression. For large time segments, consistent geometries fail to track fine scale surface details which results in inconsistencies in the projected appearances into the texture domain. Some works that consider globally consistent geometries [16], [37] address these inconsistencies on a frame basis mainly for animation synthesis through interpolation. In [38], the appearance of characters tracked with rigged sequences is animated by warping captured images selected through patch-based retrieval. All these approaches could be used to synthesize appearance within our animation pipeline, but we opt for our previous work [15], as it offers other usages such as compact reduction and missing appearance completion, besides interpolation. The abilities of this approach, that builds on previous works on static shapes [39], [40], [41], [42], stem from the fact that it constructs a compact global appearance model for a character. This model leverages and spans appearance from multiple sequences, and does not solve alignments only locally on a frame pair basis.

3 Shape pose and motion blending

In order to build animations that combine segments of the input mesh sequences, we need to generate new transitions between them. The output animation is thus a mesh sequence obtained by concatenating original mesh segments and synthetic mesh segments that ensure spatial and temporal continuity of the output motion stream. We use gradual frame blending to synthesize transitions and we explain in this Section how to do so. To this purpose, we need to define a static shape pose distance and an interpolation method. Note that we assume that all meshes have temporally consistent topology and connectivity.

3.1 Shape pose distance

While previous work tackled the problem of shape similarity analysis as in [43] between non-temporally aligned topologies, we focus here on the simpler problem of computing pose distances between two mesh poses with the same topology. To do so, we first align them rigidly using orthogonal Procrustes analysis. One naive strategy is to consider the residual of this alignment cost, which is the sum of Euclidean distances between corresponding aligned mesh vertices, as a distance between the poses. In our experiments, this Euclidean representation does not account for the mesh geometry and results in mesh shrinkage and distortion when performing linear mesh interpolation (Figure 4). We hence consider a different pose parametrization that builds on orthogonal factorization of local mesh deformations.

In any frame mesh, we denote by v_k the vector containing the k^{st} vertex coordinates $v_k = (x_k, y_k, z_k)^T$. We consider triangle $(v_1^i,\ v_2^i,\ v_3^i)$ in mesh i and its counterpart $(v_1^j,\ v_2^j,\ v_3^j)$ in mesh j and we append both triangles with their unit normals n_i and n_j respectively. We compute the affine transformation T that maps the first local triangle coordinate frame to the second as (Figure 4):

$$T = \left(v_1^i - v_3^i,\ v_2^i - v_3^i,\ n^i\right)^{-1} . \left(v_1^j - v_3^j,\ v_2^j - v_3^j,\ n^j\right) \quad (1)$$

Among affine transformation components, i.e. rotation, shearing and scaling, rotations are distorted under linear matrix interpolation. Thus, extracting them properly is crucial for rigid deformation parametrization. Since rotation matrices satisfy orthogonality, extracting them can be achieved with orthogonal decomposition algorithms for non-singular 3×3 matrices. Among the methods proposed for these decompositions, the orthogonal factors of the *SVD* decomposition are not unique, and the orthogonal component in the *QR* decomposition is not independent of the coordinate basis. On the contrary, *Polar decomposition* $T = R.S$ gives a symmetric component S and an orthogonal component R that is unique, independent of the basis and stable under perturbations. We consider therefore this method for our triangle-based transformation factorization.

Since the shearing/scaling component of the polar decomposition S is positive-semidefinite, and the rotation component R belongs to the special orthogonal group, we use the Riemannian metric of the Cartesian product of these two Riemannian manifolds over all mesh triangles to define a pose similarity metric between meshes i and j:

$$d(i,j) = \sum_m \frac{1}{\sqrt{2}} \left(\| \log\left(R_m\right) \|_F + w_m \| \log\left(S_m\right) \|_F \right) \quad (2)$$

where $\|.\|_F$ is the Frobenius norm and index m covers all mesh triangles. Through weighting factors w_m, we give more importance to the rotation term. In practice, we use the same value $w_m = 0.01$ for all mesh triangles.

Evaluation

	Pearson Corr.		Distance Corr.	
	Eucl.	Ours	Eucl.	Ours
Actor 1	0.8078	0.9222	0.8031	0.9232
Actor 2	0.8749	0.9189	0.8686	0.8977
Actor 3	0.8804	0.9075	0.8582	0.8989

TABLE 1: Correlation between the ground-truth and both our pose distance and the Euclidean pose distance.

We evaluate our mesh-based pose metric empirically using the *Adobe* dataset [44]. This dataset contains various motions performed by 3 different actors in the form of coherent mesh sequences. The models represent detailed meshes of loosely dressed humans which makes this set of data relevant to our context. These sequences are additionally equipped with pose information in the form of Euler angle values of a template skeleton for each frame. We first compute distances between all pose pairs in the dataset using a metric derived from the Euclidean distance between Euler Angles [45] and consider these as ground-truth distances. We then compute the same distances using our non-euclidean pose metric (Equation 2) and a straightforward metric based on Euclidean distances between mesh vertices. We assess the relationship between the ground-truth and both the Euclidean and non-Euclidean distances by evaluating both linear and non-linear dependencies using Pearson Correlation Coefficient and Distance Correlation [46] respectively. Results in Table 1 show that our pose metric consistently better correlates to the ground-truth than the naive vertex based Euclidean distance for both correlation measures.

3.2 Shape pose interpolation

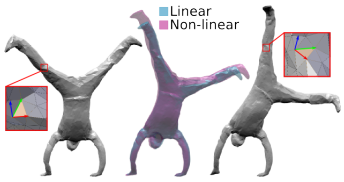


Fig. 4: Linear vs. Non-linear pose interpolation.

In order to perform interpolations between frames that preserve local mesh properties better than naive interpolation we use a variant of Poisson shape interpolation [47]. Note that other alternatives might be considered such as As-rigid-as-possible approaches [48]. In differential triangular mesh processing [13], expressing the mesh as a piece-wise linear coordinate function results in constant mesh gradients within each triangle. This 3×3 per-face gradient of the mesh for triangle m can be expressed as follows:

$$G_m = \begin{pmatrix} (v_1 - v_3)^T \\ (v_2 - v_3)^T \\ n^T \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ v_3^T \end{pmatrix} \tag{3}$$

Denoting by N and M the numbers of mesh vertices and triangles respectively, a global gradient operator G can be yielded in the form of a $3M \times N$ matrix that multiplies the stacked transposed mesh vertex coordinates in a $N \times 3$ matrix, to give a $3M \times 3$ matrix of stacked triangle gradients:

$$\begin{pmatrix} G_1 \\ \vdots \\ G_M \end{pmatrix} = G \begin{pmatrix} v_1^T \\ \vdots \\ v_N^T \end{pmatrix} \tag{4}$$

To interpolate poses with an interpolation parameter $\lambda \in [0,1]$, and for every triangle m in the mesh, we first interpolate the rotation component R_m and the shearing/scaling component S_m geodesically in the special orthogonal group and the manifold of positive semi-definite 3×3 matrices respectively:

$$\tilde{T}(\lambda) = \exp\left(\lambda \log(R_m)\right) \exp\left(\lambda \log(S_m)\right) \tag{5}$$

We then deform each triangle based gradient G_m in the source mesh by the interpolated local transformation $\tilde{T}_m$ to obtain the new gradients $\tilde{G}_m$:

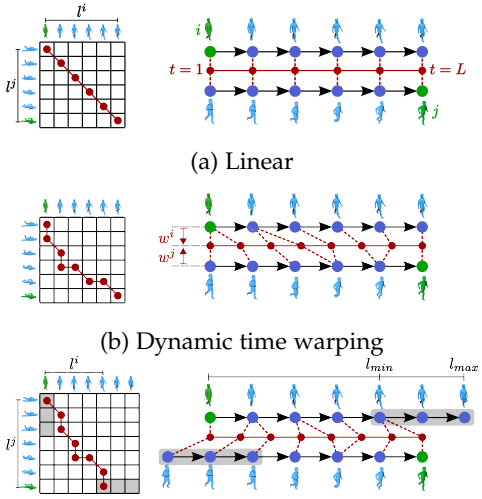
$$\tilde{G}_m = \tilde{T}_m G_m \tag{6}$$

These new deformed gradients are used to reconstruct the interpolated mesh composed of vertices $\{\tilde{v}_1, ..., \tilde{v}_N\}$. This resulting surface consists of the vertex sites matching the deformed gradients in Equation 6, and is found by solving the following linear sparse Poisson system:

$$\underbrace{G^T D G}_{\Delta} \begin{pmatrix} \tilde{v}_1^T \\ \vdots \\ \tilde{v}_N^T \end{pmatrix} = \underbrace{G^T D}_{\vec{\nabla}\cdot} \begin{pmatrix} \tilde{G}_1 \\ \vdots \\ \tilde{G}_M \end{pmatrix} \tag{7}$$

where D is a diagonal $3M \times 3M$ mass matrix containing the mesh triangle areas. To ensure the uniqueness of the final interpolated mesh, we deform both the source a target meshes towards each other using this method, and the two results are then interpolated linearly.

3.3 Shape motion transition



(a) Linear

(b) Dynamic time warping

(c) Dynamic time warping + Varying length segments

Fig. 5: Gradual frame blending transition optimization.

In order to generate a smooth interpolated transition from frame i in a source sequence to frame j in a destination sequence, we consider a window of successive frames of size l^i in the source sequence starting at frame i: $[i, i + l^i - 1]$, and a similar window of successive frames of size l^j in the destination sequence ending at frame j: $[j - l^j + 1, j]$. These are the source and destination segments whose frames are to be blended gradually so as to generate a smooth transition between frames i and j, in a similar way to a standard linear interpolated transition (Figure 5a), such as

the solution proposed for fixed sized windows in [17]. This interpolation can be first improved by temporally aligning source and destination motion segments with warps w^i and w^j respectively (Figure 5b). We use a standard dynamic time warping algorithm [49] to compute the best temporal warps, where the cost matrix is built using the pose distance in Equation 2. In addition, following [31] who argued that a transition duration in skeletal motion interpolation should be comprised between a third of a second and 2 seconds, we can also allow segment lengths l^i and l^j to vary between boundaries l_{min} and l_{max} that correspond to these values (Figure 5c). We can finally choose the pair (l^i, l^j) with the least cost through the dynamic time warping path:

$$D(i,j) = \min_{l^i,l^j} \min_{w^i,w^j} \sum_{t\in[\![0,L]\!]} d(w^{i^{-1}}(t), w^{j^{-1}}(t)) \qquad (8)$$

We refer to this cost in the following as the surface deformation cost of a transition $D_{i,j} = D(i,j)$. The length of the dynamic time warping path gives the length of the transition, which defines the transition duration cost $L_{i,j}$.

Evaluation

We evaluate in Figure 7 our optimized interpolated transitions numerically using datasets TOMAS [14], DAN [16] and JP [50], [51]. We compute the average transition cost between all possible pairs of frames in these datasets. In Tables 7a, 7b, 7c, the normalized surface deformation costs that are minimized when optimizing transitions are given. For a more independent and fair metric, we also give the motion acceleration cost in Tables 7d, 7e, 7f. We define this latter cost as the average difference between accelerations computed at each vertex for every pair of source and target meshes in the dynamic time warping path, in an attempt to capture the naturalness of the synthesized resulting transition. These costs are computed for: standard interpolated transitions; transitions that use dynamic time warping only; transitions that use a combination of dynamic time warping and varying length source and target segments. When dynamic time warping is involved, we evaluate cost statistics for various slope constraints, allowing between 2 and 5 consecutive horizontal or vertical nodes in a row in the dynamic time warping path. Overall, we can see that dynamic time warping improves the transition cost in average and this improvement increases when source and target segment lengths are optimized as well. We notice also that relaxing the slope constraint results in lower costs. However, allowing for a large number of vertical or horizontal nodes in a row in a dynamic time warping path might result in temporally degenerate transitions. We hence opt for an empirical limit of 3 in our synthesis.

Fig. 6: Transition from *Walk* to *Run* sequences of TOMAS. Red frames are synthetic and green frames are original frames.

We also show qualitative comparisons between standard linearly interpolated transitions and our optimized transitions in the accompanying video, Section *Shape motion transition*. These results show that our method tends to avoid surface shrinkage, foot-skate and other unnatural looking motion artifacts thanks to the improved alignment of poses in motion segments prior to interpolation. Figure 6 shows a transition synthesis example from *Walk* to *Run* movements of TOMAS dataset using our method.

4 SHAPE ANIMATION SYNTHESIS

Our example-based animation scheme consists in combining original motion bits and interpolated transitions between them. We hence need a data structure to select and encode transitions between input sequences and control the animation process. We propose in this Section a data structure solution for animation synthesis, and we explain how this structure can be used to transform a set of high-level directives into a spatially and temporally continuous mesh sequence.

4.1 Motion data structure

We use a directed weighted graph structure to organize our input mesh sequences. Nodes represent frames and edges stand for possible transitions between them. Our goal is to synthesize optimal motions from a set of example sequences with respect to a realism criterion as defined in Section 4.1.2. Hence, edge weights in the graph, that we note $E_{i,j}$ for an edge linking node i to j, represent the realism of transition segment (i,j) evaluated with the same criterion.

4.1.1 Essential graph

In a standard Motion graph [18], edges between sequences are obtained by selecting minima in transition cost matrices between pairs of sequences. This intuitive strategy is not yet globally optimal since a globally optimal path in the graph can include transitions between pairs with non minimal costs. We propose therefore a global and principled strategy that consists of extracting the best paths between any pair of poses and to keep only edges in the graph that contribute to these paths.

Figure 8 illustrates how an Essential graph is built. First, the initial instance of the graph is drawn from the input dataset. Hence, edges only connect successive frames in the original sequences. In the second stage, we connect all nodes with directed weighted edges, thus forming a complete digraph. In our implementation, the edge weighing function $E_{.,.}$ satisfies positivity and positive definiteness. It is however lacking symmetry and triangle inequality to define a proper metric. Hence, the resulting graph structure is analogous to a quasi-semi-metric space, (V,E), where V is the set of nodes and $E : V \times V \rightarrow \mathbb{R}^+$ is the asymmetric function that defines edge weight values. Finally, for every pair of nodes in the complete graph, we use Dijkstra algorithm to find the path with the least cost joining the source node to the destination node. The cost of a path $p = [n_1, n_2, ..., n_N]$ that goes through nodes $n_1, n_2, ..., n_N$

			Mean	Std. dev.
Linear			0.2490	0.0758
DTW	Slope	2	0.2446	0.0769
		3	0.2427	0.0772
		4	0.2415	0.0774
		5	0.2409	0.0774
DTW+VLS	Slope	2	0.1942	0.0607
		3	0.1860	0.0574
		4	0.1817	0.0557
		5	0.1789	0.0547

(a) Normalized surface deformation transition cost, TOMAS

			Mean	Std. dev.
Linear			0.2231	0.0706
DTW	Slope	2	0.2184	0.0713
		3	0.2161	0.0717
		4	0.2147	0.0719
		5	0.2138	0.0719
DTW+VLS	Slope	2	0.1731	0.0550
		3	0.1662	0.0519
		4	0.1629	0.0502
		5	0.1609	0.0490

(b) Normalized surface deformation transition cost, DAN

			Mean	Std. dev.
Linear			0.4855	0.1626
DTW	Slope	2	0.4827	0.1630
		3	0.4817	0.1631
		4	0.4813	0.1632
		5	0.4812	0.1632
DTW+VLS	Slope	2	0.4043	0.1560
		3	0.3941	0.1543
		4	0.3893	0.1532
		5	0.3865	0.1525

(c) Normalized surface deformation transition cost, JP

			Mean	Std. dev.
Linear			0.1933	0.0410
DTW	Slope	2	0.1909	0.0432
		3	0.1898	0.0440
		4	0.1892	0.0444
		5	0.1889	0.0446
DTW+VLS	Slope	2	0.1776	0.0462
		3	0.1774	0.0448
		4	0.1767	0.0432
		5	0.1762	0.0420

(d) Normalized acceleration transition cost, TOMAS

			Mean	Std. dev.
Linear			0.2448	0.0462
DTW	Slope	2	0.2432	0.0472
		3	0.2425	0.0477
		4	0.2417	0.0479
		5	0.2412	0.0482
DTW+VLS	Slope	2	0.2482	0.0651
		3	0.2444	0.0600
		4	0.2396	0.0540
		5	0.2346	0.0486

(e) Normalized acceleration transition cost, DAN

			Mean	Std. dev.
Linear			0.5152	0.1510
DTW	Slope	2	0.5134	0.1508
		3	0.5131	0.1507
		4	0.5130	0.1507
		5	0.5130	0.1507
DTW+VLS	Slope	2	0.4860	0.1552
		3	0.4854	0.1537
		4	0.4845	0.1516
		5	0.4830	0.1488

(f) Normalized acceleration transition cost, JP

Fig. 7: Comparison of linear transitions, transitions with dynamic time warping (DTW), and transitions with dynamic time warping and varying length segments (DTW+VLS).

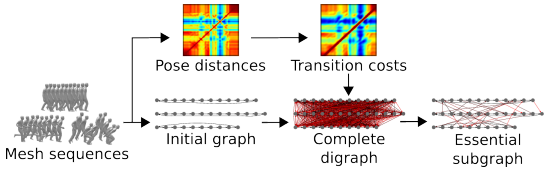


Fig. 8: Building the essential graph.

is defined as the sum of the costs of all edges forming this path sequentially:

$$J(p) = \sum_{i \in [\![1, N-1]\!]} E_{n_i, n_{i+1}} \tag{9}$$

Once the optimal paths joining all pairs of nodes are found, all edges that do not belong to any of these paths are pruned. The resulting structure is also referred to as the union of shortest path trees rooted at every graph node.

4.1.2 Graph edge weights

Weights attributed to edges in our graph should ideally capture the realism of the transitions they represent. Following the seminal work [52], we quantify this realism with a criterion that derives from both physical and temporal considerations. It takes into account a surface deformation cost that we estimate using cost function $D(.,.)$ in Equation 8, that measures the surface interpolation cost of a synthetic transition, and we set this cost to zero for original edges. The realism criterion takes also into account a duration cost that accounts for the number of frames of a motion segment, $L(.,.)$.

For each pair of nodes i and j in the graph, we define the weight $E_{i,j}$ of the directed edge (i,j) as the weighted sum of the optimal surface deformation cost $D_{i,j} = D(i,j)$ and the associated duration cost $L_{i,j}$:

$$E_{i,j} = D_{i,j} + \alpha L_{i,j} \tag{10}$$

In the situation where frames i and j belong to the same sequence, with $i < j$, it is unnecessary to consider an interpolated transition if its cost is greater than $\alpha(j-i)$, that is the cost of going from i to j through the original motion sequence.

The weight α controls flexibility on the admissible surface deformation during a transition with respect to time duration. It controls the permissibility of adding new edges in the Essential graph and hence the density of this latter. We note that the density of a directed graph is defined as the ratio of the number of its edges to the maximum number of edges. In Figure 9, and using datasets TOMAS (9d) , JP (9f) and DAN (9e), we constructed Essential graphs with various values of α and noticed that their densities increased almost linearly with the value of α. In practice for our synthesis experiments, we use a value of $\alpha = 0.01$, which seems to result in a good compromise between surface deformation and duration costs in terms of perceptual results.

Evaluation

We validate in this Section our data structure, namely the Essential graph, by comparing it to a standard Motion graph. We show quantitative comparisons in Figure 9 and additional qualitative comparisons in the accompanying video, Section *Motion data structure*.

Given a dataset of mesh sequences, we want to generate motions joining all frames and minimizing a joint cost of surface deformation and duration (Equation 10). To this end, we construct an Essential graph and a Motion graph, and compute the costs of optimal paths joining every pair of frames in the dataset. In order to compare graph structures independently of the transition approach, we use the same transition cost for both methods. To construct a Motion graph, we compute transition cost matrices between all pairs of sequences and select the local minima within them as transition points. In the original paper [18] implementa-

α	Motion Graph		Essential Graph	
	Mean	Std. dev.	Mean	Std. dev
0.01	2.0696	0.6040	2.0366	0.5985
0.02	2.3072	0.6203	2.2619	0.6168
0.03	2.5232	0.6375	2.4583	0.6331
0.05	2.9201	0.6800	2.8008	0.6694
0.1	3.8128	0.8158	3.4756	0.7406
0.2	5.4628	1.2008	4.5102	0.7917

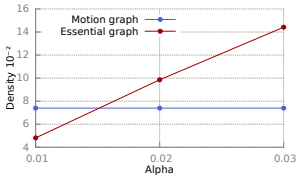
(a) Average shortest path costs, TOMAS

α	Motion Graph		Essential Graph	
	Mean	Std. dev.	Mean	Std. dev
0.01	1.5589	0.4757	1.5509	0.4753
0.02	1.7595	0.4782	1.7392	0.4785
0.03	1.9391	0.4844	1.9006	0.4858
0.05	2.2695	0.5069	2.1830	0.5077
0.1	3.0397	0.6043	2.7847	0.5759
0.2	4.5011	0.8927	3.8083	0.6837

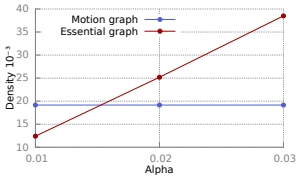
(b) Average shortest path costs, DAN

α	Motion Graph		Essential Graph	
	Mean	Std. dev.	Mean	Std. dev
0.01	2.2705	0.3990	2.2272	0.4002
0.02	2.7922	0.6676	2.7184	0.6619
0.03	3.2526	0.9098	3.1367	0.8999
0.05	4.0316	1.2635	3.8090	1.2454
0.1	5.4115	1.6108	4.8709	1.5476
0.2	7.5315	2.0157	6.2073	1.7041

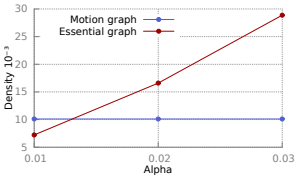
(c) Average shortest path costs, JP



(d) Graph density, TOMAS



(e) Graph density, DAN



(f) Graph density, JP

Fig. 9: Comparison between Motion graphs and Essential graphs.

tion, it is also recommended to only accept local minima below an empirically determined threshold to reduce the computational burden. We deliberately ignore this step to obtain a highly connected Motion graph to better challenge our method. Note that we do not compare to existing extensions of the Motion graph since our primary objective is to evaluate the initial structure that is used to select optimal transitions, and we believe that many of these extensions could anyway apply to the Essential graph as well. We reiterate these experiments for the three dataset TOMAS, DAN and JP and for different values of the α parameter weighting surface deformation to duration cost.

Figures 9a, 9b and 9c show the mean and the variance of the shortest path costs between all frame pairs in TOMAS, DAN and JP datasets respectively for both the Essential graph and the Motion graph, and for different values of α. As illustrated in these figures, the Essential graph gives shortest paths with smaller costs than the Motion graph in average for all our datasets. The accompanying video illustrates in its Section *Motion data structure* cases where a motion graph fails to find a graph walk connecting a source and target frames that is as short and costless as the one found by the essential graph. One could suspect that our graph representation outperforms a standard Motion graph numerically because of a higher graph density, as a graph with more edges allows for more flexibility in path optimization. The results in Figures 9d, 9e and 9f show that this hypothesis is not true: for a value of $\alpha = 0.01$ for instance, Essential graphs are less dense than Motion graphs for the three datasets TOMAS, DAN and JP. Nevertheless, Figures 9a,9b and 9c still show a lower average shortest path cost for the Essential graphs compared to Motion graphs in this case. Hence, with a more sparsely connected graph, our method still performs numerically better than standard Motion graphs. This proves that the reason behind the good performance of our method is not the density of our graph but rather its *better* connectivity.

We also notice in Figures 9a, 9b and 9c that for smaller values of α, the Essential graph converges in its shortest path costs towards the Motion graph. As a matter of fact, building the Essential graph with $\alpha = 0$ means discarding the duration cost in the numerical realism criterion, which is equivalent to solving the optimal transition selection problem in the case where initial edges between original sequence nodes are cost-less. In this specific case, we know that the Motion graph covers the optimal solution by definition, and hence it coincides with the Essential graph.

The performance of our method comes however with a significant computational complexity. We denote in the following by V the set of graph nodes, and by E the set of graph edges. The complexity of a simple implementation of Dijkstra algorithm is $\mathcal{O}(|E| + |V|^2)$ which equals $\mathcal{O}(|V|^2)$ since $|E| = \mathcal{O}(|V|^2)$, where operator $|.|$ symbolizes the cardinality of a set. In practice, we only need to find the shortest paths between all pairs of nodes in the complete digraph in order to build the Essential graph. Hence, the final complexity of building the Essential graph naively amounts to $\mathcal{O}(|V|^4)$. However, we are not highly concerned with this cost since this step is performed off-line in our animation framework, and mesh motion datasets available so far are relatively small in size.

4.2 Motion synthesis

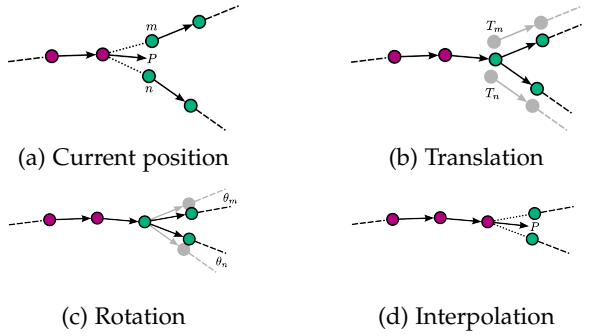
Motion synthesis consists in converting a walk in the Essential graph into a temporally and spatially continuous motion stream in the form of a 3D mesh sequence. A Graph walk path is represented by a node sequence, where every pair of consecutive nodes must be linked by an edge in the Essential graph. We browse the path and sequentially identify two types of motion segments: any succession of nodes belonging to an input sequence in their original order forms an *original segment*. When a non original transition appears in the walk path, it signals the end of the current original segment. The next segment consists then of this interpolated transition frames, and we refer to it as a *synthetic segment*.

A synthetic segment represents a transition from a frame i to a frame j in two sequences of the dataset. Such segment is generated using the optimal parameters of the interpolated transition introduced in Section 3.3, namely source segment length l^i and temporal warp w^i, destination segment length l^j and temporal warp w^j and transition length L. We blend source segment $[\![i, i + l^i - 1]\!]$ warped by w^i with destination segment $[\![j - l^j + 1, j]\!]$ warped by w^j

Data: Source segment $[\mathcal{M}_i, \mathcal{M}_{i+l^i-1}]$, Destination segment $[\mathcal{M}_{j-l^j+1}, \mathcal{M}_j]$, Source warp w^i, Destination warp w^j, Transition duration L.
Result: Output segment S_{out}
Current mesh center of mass $P := \bar{\mathcal{M}}_i$;
for $t \in [\![1, L]\!]$ **do**
  Current source frame $n := {w^i}^{-1}(t)$;
  Current destination frame $m := {w^j}^{-1}(t)$;
  $\lambda := (t-1)/(L-1)$;
  $\theta := \text{Align}(\mathcal{M}_n, \mathcal{M}_m)$;
  $\theta_n := -\lambda\theta$;
  $\theta_m := (1-\lambda)\theta$;
  $T_n := P - \bar{\mathcal{M}}_n$;
  $T_m := P - \bar{\mathcal{M}}_m$;
  $\text{Move}([\mathcal{M}_n, \mathcal{M}_{i+l^i-1}], \theta_n, \bar{\mathcal{M}}_n, T_n)$;
  $\text{Move}([\mathcal{M}_m, \mathcal{M}_j], \theta_m, \bar{\mathcal{M}}_m, T_m)$;
  $\mathcal{M}_t := \text{Interpolate}(\mathcal{M}_n, \mathcal{M}_m, \lambda)$;
  $P := P + (1-\lambda)(\bar{\mathcal{M}}_{n+1} - \bar{\mathcal{M}}_n) + \lambda(\bar{\mathcal{M}}_{m+1} - \bar{\mathcal{M}}_m)$;
  $S_{out} := [S_{out}, \mathcal{M}_t]$;
end

Algorithm 1: function Blend()

with a parameter that evolves continuously to ensure the smoothness of the generated segment.



(a) Current position (b) Translation

(c) Rotation (d) Interpolation

Fig. 10: One step of the recurring 2D alignment algorithm for motion segment blending. Red nodes are blended and green ones are original.

Motion segment blending is described in Algorithm 1. The function detailed in this algorithm, that we call Blend(), takes as input source and destination segments with their respective alignment temporal warps, and outputs the interpolated transition segment. In this algorithm, function $\text{Move}(S, \theta, C, T)$ moves sequence of meshes S by a rotation around the perpendicular axis to the motion plane, where θ is the rotation angle and C the rotation center. It also applies a translation T to the mesh sequence. Function $\text{Align}(\mathcal{M}_1, \mathcal{M}_2)$ finds the angle by which mesh $\mathcal{M}_2$ needs to be rotated to align with mesh $\mathcal{M}_1$. Function $\text{Interpolate}(\mathcal{M}_1, \mathcal{M}_2, \lambda)$ interpolates meshes $\mathcal{M}_1$ and $\mathcal{M}_2$ non-linearly.

At each step of the process, illustrated also in Figure 10, the current source and destination frame indices n and m are obtained through inverse source and destination time warps w^i and w^j. Next, we move the current remaining parts of source segment $[\mathcal{M}_n, \mathcal{M}_{i+l^i-1}]$ and destination segment $[\mathcal{M}_m, \mathcal{M}_j]$ with planar translations T_n and T_m towards P the current position of mesh's center of mass

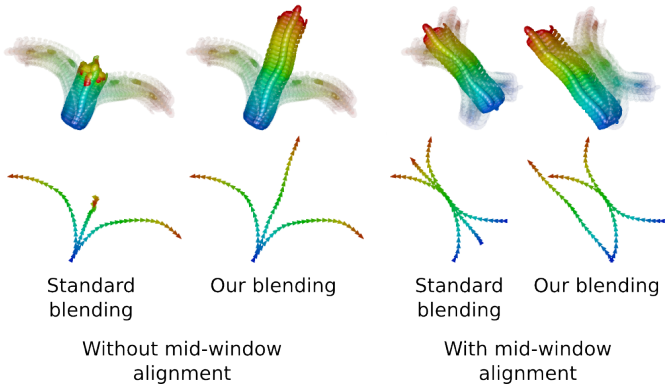


Fig. 11: Blending sequences *Left* and *Right* of TOMAS, constant blending weight of 0.5.

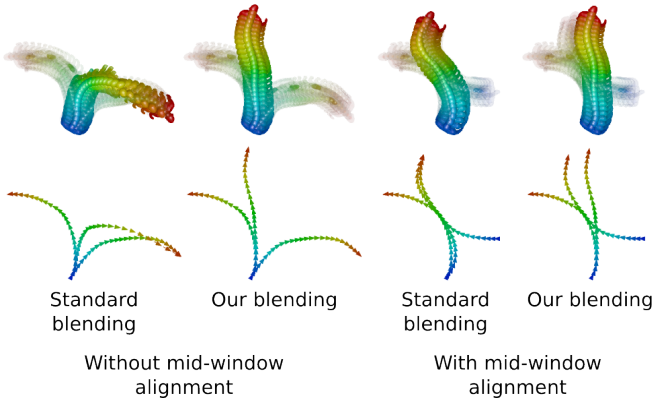


Fig. 12: Blending sequences *Left* and *Right* of TOMAS, increasing blending weight from 0 to 1.

(Figure 10b). We also align them with respect to the their first frames $\mathcal{M}_n$ and $\mathcal{M}_m$ with rotations θ_n and θ_m respectively around the vertical axis (Figure 10c). After that, current meshes $\mathcal{M}_n$ and $\mathcal{M}_m$ are interpolated non-linearly, and the current source and destination displacements of center of mass are also interpolated, thus giving the next mesh center of mass position (Figure 10d).

In practice, our motion segment blending algorithm differs from standard linear blending with its recurring 2D rigid alignment step of current source and destination segments, which is somehow similar to coordinate alignment in the work of [30] on registration curves for motion capture data. Figures 11 and 12 illustrate the impact of this process compared to a standard blending algorithm, such as the strategy proposed in [17]. In both figures, we show the result of blending sequences *Left* and *Right* from TOMAS dataset with both meshes at the top of the figures, and their corresponding center of mass trajectory projected on the motion plane in the bottom. In their algorithm, [17] recommend to pre-align the input sequences with respect to the central frame of the window prior to blending, hence we show comparisons with and without this mid-window alignment.

Overall, our algorithm prevents the trajectory of center of mass of the synthesized sequence from collapsing or reversing path. This allows us to synthesize more natural looking motion transitions without major visual flaws

and unnatural character global trajectories, regardless of the nature of the motions to be blended. These benefits are better visualized in the case of blending with constant weight (0.5) as shown in Figure 11. Without mid-window alignment, the center of mass trajectory in [17] reverses path. Although the alignment improves this aspect, the final trajectory in this case is still shorter than ours which points out unnatural motion contraction. In Figure 12, we blend the input sequences with a gradually increasing weight from 0 to 1. Even without pre-alignment, our method generates smoother and more natural looking transition trajectories that do not change direction abruptly.

Data: Graph walk path $p := [n_1, \dots, n_N]$
Result: Output mesh sequence S_{out}
current rotation angle θ_c, current rotation center C_c, current translation T_c
for $i \in [\![1, N-1]\!]$ **do**
  if $[\![n_i, n_{i+1}]\!]$ *is an original segment* **then**
    Move($[\mathcal{M}_{n_i}, \mathcal{M}_{n_{i+1}}], \theta_c, C_c, T_c$);
    $S_{out} := [S_{out}, [\mathcal{M}_{n_i}, \mathcal{M}_{n_{i+1}}]$[;
  end
  else
    Move($[\mathcal{M}_{n_i}, \mathcal{M}_{n_i + l^{n_i} - 1}], \theta_c, C_c, T_c$);
    $S := \text{Blend}([\mathcal{M}_{n_i}, \mathcal{M}_{n_i + l^{n_i} - 1}],$
      $[\mathcal{M}_{n_{i+1} - l^{n_{i+1}} + 1}, \mathcal{M}_{n_{i+1}}])$;
    $S_{out} := [S_{out}, S$[;
    $\theta_c := \text{Align}(S(\text{end}), \mathcal{M}_{n_{i+1}})$;
    $C_c := \bar{\mathcal{M}}_{n_{i+1}}$;
    $T_c := \bar{S}(\text{end}) - \bar{\mathcal{M}}_{n_{i+1}}$;
  end
end
Move($\mathcal{M}_{n_N}, \theta_c, C_c, T_c$);
$S_{out} := [S_{out}, \mathcal{M}_{n_N}]$;

Algorithm 2: Algorithm to generate a motion sequence from a graph walk

Finally, Algorithm 2 details how motion sequences are generated from a general walk in the Essential subgraph. The final sequence is mainly a concatenation of original and synthetic motion segments. In order to ensure spatial continuity of the resulting motion stream, we additionally perform 2D rigid alignment at the junction between two consecutive motion segments.

4.3 High-level constraints

As generating random graph walks has limited practical interests, we consider in this part constrained navigations in the Essential graph with various user-specified constraints such as spatial, temporal and behavioural constraints. We cast motion extraction as a graph search problem, that consists in finding the walk $p = [n_1, n_2, ..., n_N]$ that minimizes a total error cost $J_c(p)$ defined as follows:

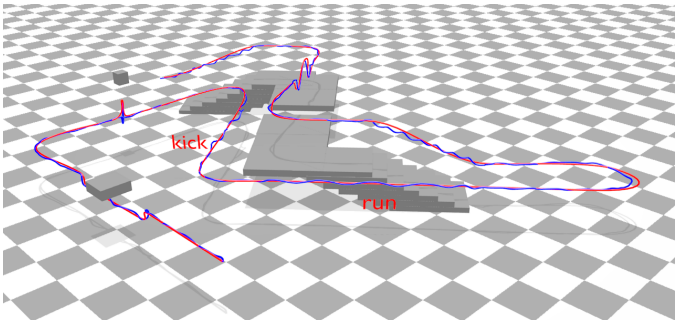
$$J_c(p) = \sum_{i \in [\![1,N]\!]} j_c([n_1, ..., n_{i-1}], n_i) \qquad (11)$$

where $j_c([n_1, ..., n_{i-1}], n_i)$ is a scalar function evaluating the additional error of appending node n_i to the graph walk $[n_1, ..., n_{i-1}]$ with respect to the user specification. Since a graph node might be visited more than once in this search, the user also needs to specify a halting condition to prevent the search from recursing infinitely.

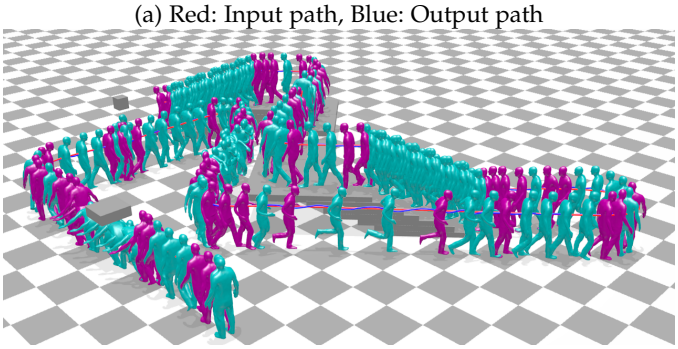
In order to demonstrate the interest of Essential graphs for animation purposes, we implement two scenarios with different types of constraints. In both scenarios, the graph search was solved using Depth-First algorithm with Branch-and-Bound to reduce the search space. When both the start and end nodes are known as in the dance sequence editing scenario in Section 4.3.2, we use bi-directional search for a faster graph search.

4.3.1 3D path synthesis

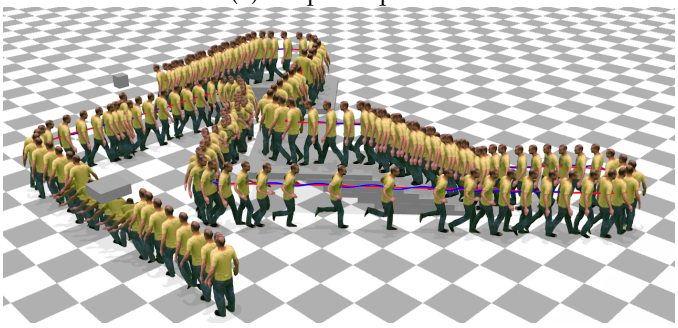
Given a set of motion sequences of the same character, the user provides a 3D curve that the character's center of mass should follow as closely as possible. Additionally, some parts of this path may impose specific types of motion taken from the dataset.



(a) Red: Input path, Blue: Output path



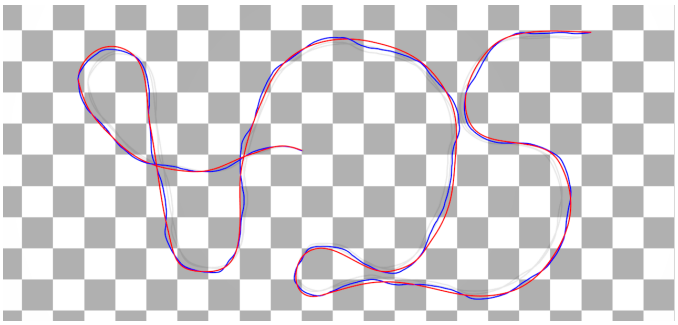
(b) Output sequence



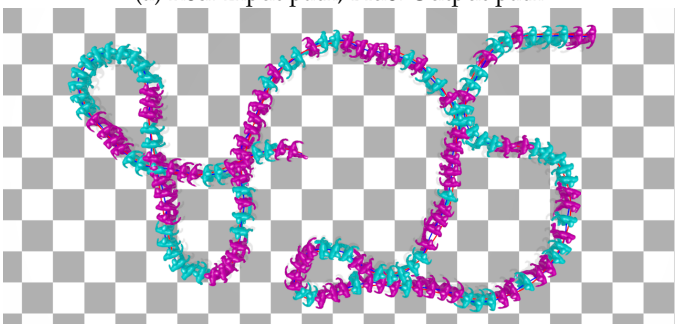
(c) Textured output sequence

Fig. 13: 3D path synthesis.

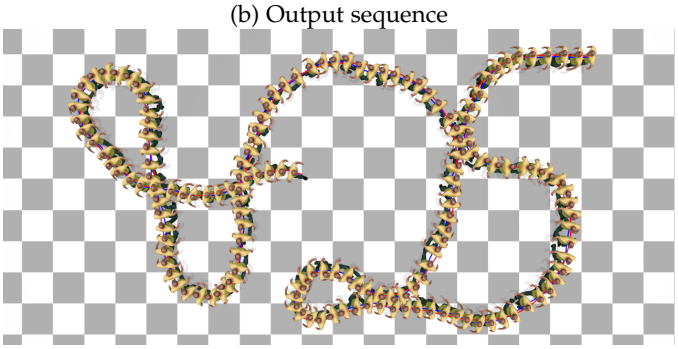
We test this scenario using TOMAS dataset. We show in Figure 13 and the accompanying video (Section *3D path synthesis*) an example where the character has to follow a 3D curve provided by the user. For some segments of the curve, the user also requires that the character performs



(a) Red: Input path, Blue: Output path



(b) Output sequence



(c) Textured output sequence

Fig. 14: 2D path synthesis.

Kicking and *Running* motions exclusively. Figure 14 and the accompanying video (Section *2D path synthesis*) show another example where we only require the projection of the center of mass of the character to follow a 2D path in the motion plane. In both examples, green output frames are original ones, and red frames are interpolated ones. We also show the output sequence with textured meshes.

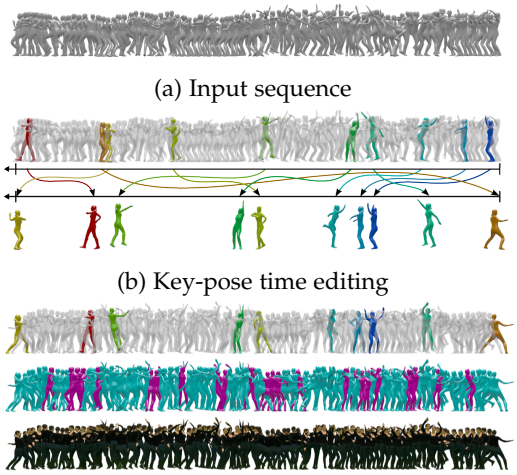
4.3.2 *Pose/time constraints*

In this example scenario, and given an input motion sequence, the user selects a set of key-poses and provides new times of occurrence for each of them. The graph search generates the motion sequence that respects as closely as possible these pose/time restrictions.

Input	1480	464	21	119	275	1212	541	864	1762	1459
Output	0	250	460	510	610	890	970	1420	1520	1800
Result	0	258	460	514	617	886	970	1433	1530	1797

TABLE 2: CATY dance sequence editing results.

In Figure 15, we show an example where we apply this scenario to synthesize a new variant of the dance



(a) Input sequence

(b) Key-pose time editing

(c) Output sequence

Fig. 15: Pose/time editing.

sequence in CATY dataset, where certain key-poses appear at different timings from their original occurrence times in the input sequence. In this Figure, green output frames are original ones, and red frames are interpolated ones. We also show the output mesh sequence with textures. Table 2 shows the numerical results of this experiment: in the top row, we show the original occurrence times of the selected key-poses. In the second row we show the desired times of occurrence of the same key-poses in the output sequence. Finally in the last row, we show the real times of occurrence of these key-frames in the output sequence that we obtained with the minimal cost walk. Notice that for the most part, the obtained times of occurrence are very close and sometimes equal to the desired times. The timing errors are not propagated as graph search is done sequentially for every successive pair of start and end poses, and the timing error in the previously solved pair is taken into account in the next pair search. As we show also in the accompanying video, Section *Pose/time constraint*, the resulting dance sequence nearly matches the input sequence in realism and the transitions are visually plausible.

5 Appearance Animation Synthesis

In order to append appearance to shape animation, we need to synthesize appearance for the interpolated shapes. Interpolating texture maps linearly for that purpose results in Ghosting artifacts as shown in Figure 21. These artifacts are a result of texture misalignments due to inaccuracies in the geometric modelling process. Since improving the underlying geometry is a very challenging task, we address this problem by proposing an appearance representation of dynamic shapes that collects appearance information from all views and time instants, and compensates for the geometric inaccuracies in UV domain with a set of vectors denoted Eigen warps, in addition to model aligned appearance variation due to change in illumination and views through a set of vectors denoted Eigen textures. These two sets of Eigen vectors are referred to as Eigen appearance maps, and when combined they allow for a compact representation of texture information, and other applications

such as interpolation and completion. We briefly remind in the following how this representation is built.

5.1 Eigen appearance maps

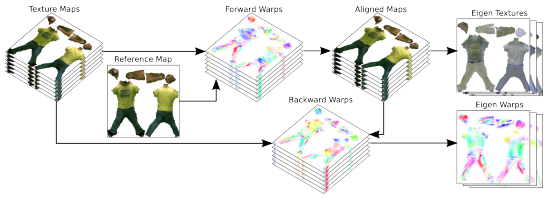


Fig. 16: Building an appearance representation from input textures (left) to Eigen maps (right).

As illustrated in Figure 16, we first select the reference texture map with the least average alignment error to all texture maps using an iterative medoid shift strategy. We then align all texture maps to this reference map and refer to the alignment warps as Forward warps. We assume that the variation in this aligned texture space can be modelled linearly and hence reduce its dimensionality through Principal Component Analysis. The Eigen components obtained from this reduction encode variation in appearance due to change in viewpoint and illumination, and we refer to them as Eigen textures. We estimate a second set of warps that map the aligned textures back to the original input textures and refer to them as Backward warps. Since these warps account for residual geometric variations, they are also advantageously decomposed using linear PCA. The Eigen vectors obtained from this second reduction encode inaccuracies in geometric modelling, and are referred to as Eigen warps. To summarize, any texture map can be represented as a linear combination of Eigen textures de-aligned with a linear combination of Eigen warps. In this pipeline, both forward and backward warps represent dense texture map pixel correspondences and are estimated using a standard optical flow implementation [53]. These various steps involved in building our appearance representation are elaborated in [15]. We evaluate the performance of this method in the following section.

5.2 Evaluation

To validate the estimation quality of our method, we apply our estimation pipeline to datasets TOMAS and CATY [15], project and warp input data using the built Eigen-spaces, then evaluate the reconstruction error. To distinguish the different error sources, we evaluate this error both in texture space before projection, and in image domain by projecting into the input views. We compare the reconstructed textures to the textures estimated from the original images in our pipeline using [12], that we consider as ground-truth textures. We also project the reconstructed textures into the input views and compare the resulting images to the original input views, that we consider as ground-truth images. For the image error measurement, we use the 3D model that was fitted to the sequence, i.e. tracked to fit the test frames selected, and render the model as textured with our reconstructed appearance map, using a standard graphics pipeline. In both cases, we experiment with both the structural similarity index (SSIM) [54] and the mean square error (MSE) as metrics to compare to the original data. All of our error estimates are computed in the active regions of the texture and image domains. In the texture domain, that means that we only consider the set of texels actually mapped to the 3D model. In the Image domain, it means that only silhouettes are taken into consideration. For image domain evaluations, we plot the average error among all viewpoints.

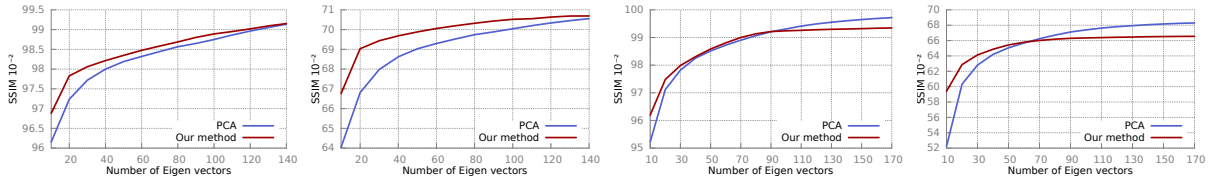
We study in particular the compactness and generalization abilities of our method, by examining the error response as a function of the number of Eigen components kept after constructing the linear subspaces, and the number of training images selected. For all these evaluations, we also provide the results for a naive PCA strategy, where only a set of Eigen appearance maps are built in texture space and used to project and reconstruct textures, to show the performance contribution of including the Eigen warps.

5.2.1 *Compactness*

For a fair comparison, we plot the results of reconstructing textures and images with our method using a total number of Eigen components that amounts to the same number of Eigen vectors used for the naive PCA on texture maps. Our Eigen maps include both Eigen textures and Eigen warps with constant proportions in the total number of Eigen components. In our experiments, we found that using two thirds of the Eigen maps for textures and the rest for warps results in nearly optimal performance.

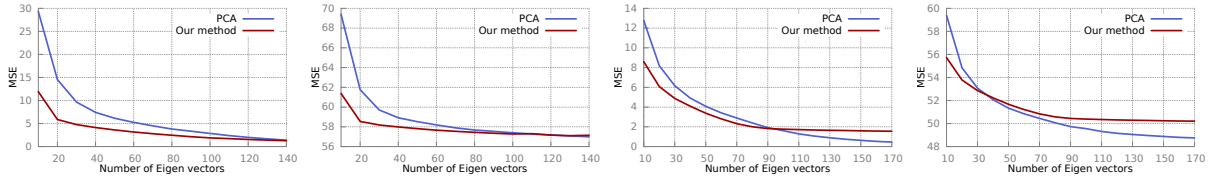
Our method outperforms naive PCA in image and texture domains on both datasets, achieving higher quality with a lower number of Eigen components, and only marginally lower quality as the number of components grows, where the method would be anyway less useful. Higher number of Eigen components marginally favours naive PCA because it converges to input textures when increasing the Eigen textures retained by construction. Because of the non bijective behaviour of texture alignment and de-alignment in our pipeline, our method hits on the other hand a quality plateau due to small constant errors. The PCA curve is hence bound to surpass ours at a given number of Eigen components.

Figure 17 shows the structural similarity reconstruction errors in both image and texture domains for CATY and TOMAS datasets. To illustrate the validity of the linear variability hypothesis in texture domain, with only 50 components, a low fraction compared to the number of input frames for both datasets (207 for TOMAS and 290 for CATY), our method introduces nearly no error (0.98 SSIM) in texture domain. The error is higher in the image domain (bounded by 0.7 in SSIM) for both our method and naive PCA, because measurements are then subject to fixed upstream errors due to geometric alignments, projections and image discretizations. Figure 18 shows the mean square reconstruction errors in both image and texture domains for CATY and TOMAS datasets. We notice that the MSE shows similar behaviour to the SSIM measure in both texture and image domains and for both datasets. The only main difference is the point where the naive PCA surpasses our method. This happens earlier at a smaller subspace dimension for the MSE curves compared to the SSIM ones, which implies that this metric



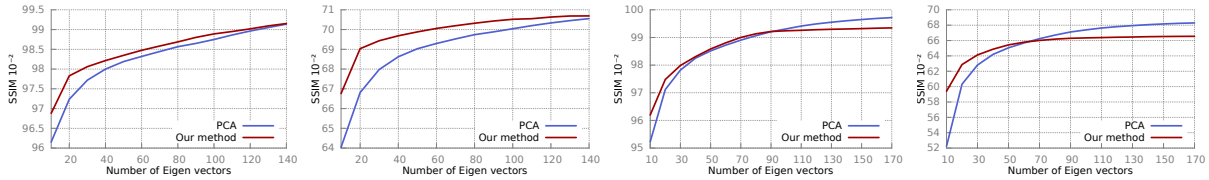
(a) Texture, TOMAS (b) Image, TOMAS (c) Texture, CATY (d) Image, CATY

Fig. 17: SSIM Reconstruction Error for TOMAS and CATY in Image and Texture domains.



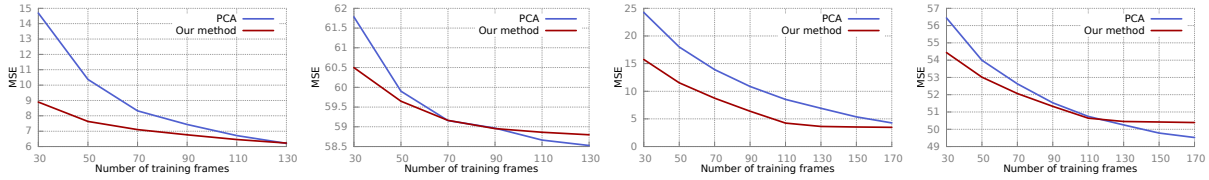
(a) Texture, TOMAS (b) Image, TOMAS (c) Texture, CATY (d) Image, CATY

Fig. 18: MSE Reconstruction Error for TOMAS and CATY in Image and Texture domains.



(a) Texture, TOMAS (b) Image, TOMAS (c) Texture, CATY (d) Image, CATY

Fig. 19: SSIM Generalization Error for TOMAS and CATY in Image and Texture domains.



(a) Texture, TOMAS (b) Image, TOMAS (c) Texture, CATY (d) Image, CATY

Fig. 20: MSE Generalization Error for TOMAS and CATY in Image and Texture domains.

favours our method less than the SSIM. We suspect this is due to the fact that, unlike the SSIM, the MSE does not take into account pixel neighbourhood information and hence is very sensitive to errors introduced with texture warping. However, structural similarity [54] was proved to be a reliable measure of visual quality and was previously used in similar evaluations involving 3D shape texture map quality assessment [36]. We hence conclude that it is more fit to evaluate the quality and the compactness of our model.

5.2.2 Generalization

In this Section, we evaluate the ability of the model to reconstruct instances outside of the training set. We use a training set comprised of randomly selected frames spanning 0% to 60% of the total number of frames, among all sequences and frames of all datasets, and plot the error of projecting the complement frames on the corresponding Eigen space (Figures 19 and 20).

Figure 19 shows the structural similarity generalization errors in both image and texture domains for CATY and TOMAS datasets. These results show that our representation produces a better generalization than naive PCA, *i.e.* less training frames needed to reconstruct texture and images of equivalent quality. Figure 20 shows the mean square generalization errors in both image and texture domains for CATY and TOMAS datasets. The MSE errors seem to follow the same pattern as the SSIM errors, except for reconstructions in the image domain, where the naive PCA catches up with the performance of our method at a smaller number of training frames. We believe this can be due to the same shortcomings of the MSE error measurement mentioned in the previous section.

5.3 Applications

In addition to reducing the appearance information of our character compactly, our appearance model is used for other applications such as completing occluded texture maps. As

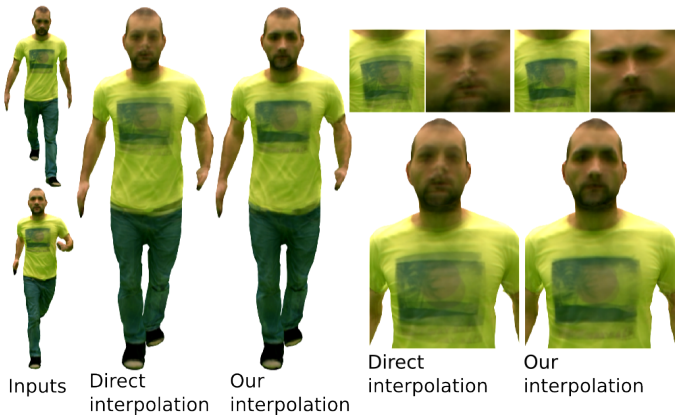


Fig. 21: Interpolating examples from TOMAS dataset using linear interpolation (left) and our pipeline (right). From left to right: Input frames, Interpolated models, and a close-up on the texture maps (top) and the rendered images (bottom).

a matter of fact, appearance maps can be incomplete due to acquisition issues. We use our appearance representation to recover the missing information from other frames and viewpoints, by omitting the incomplete maps when building the model, then re-projecting them in the Eigen spaces and reconstructing them using their projection coefficients. Another application that is very relevant to our animation pipeline is texture map interpolation. That is, when shape poses are interpolated to synthesize transitions in output mesh motion streams, their appearances need to be interpolated too to create a texture map for the interpolated pose. For the pair of input texture maps to be interpolated, we use our representation to first interpolate their corresponding warps in the Eigen warp space and aligned textures in the Eigen appearance space. The interpolated aligned texture is then de-aligned using the interpolated warp thus giving the final texture map. Our results show reduced ghosting artifacts compared to standard linear interpolation (Figure 21). More examples can be seen in the accompanying video, Sections *Appearance completion* and *Appearance interpolation*. Finally, in order to asses the impact of dimension reduction on appearance interpolation, we plot in Figure 22 the average mean square and structural similarity errors between the interpolations of TOMAS dataset's texture frame pairs, and the decompressed textures with varying texture subspace dimension under full dimensional warping. We observe that a good interpolation quality (0.99 SSIM) is reached at a relatively small number of Eigen vectors (20).

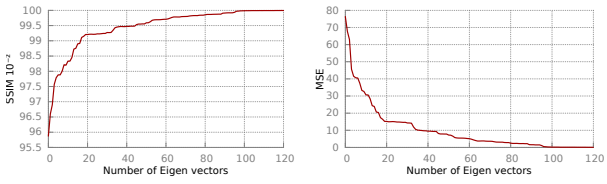


Fig. 22: The effect of texture dimension reduction on interpolation quality.

6 DISCUSSION

To build our animation pipeline, two major costly steps can be achieved offline: Building the essential graph, as explained in Section 4.1.2 , costs naively $\mathcal{O}(n^4)$ where n is the number of frames in the dataset. Building Eigen appearance maps comes to computing per frame optical flows and PCA, which costs $\mathcal{O}(n^3)$. Once the constraints are set by the user, Depth-First search for an optimal walk in our graph that has cycles remains inherently exponential even when coupled with Branch-and-Bound. Hence, in order to maintain a reasonable graph search time (about 5 seconds for 20 frames) we decompose the path to be matched to smaller segments and follow the incremental graph walk described in [18]. To render the resulting walk, non-linearly interpolated shapes can be approximated with hybrid methods [52] and hence be obtained in real time, and finally interpolating appearance maps can also be done in real time.

The main limitation of 4D animation synthesis is that the quality of the rendered animations and the performance of the appearance representation are bound by the quality of the 4D surface capture process, including surface reconstruction, tracking and appearance reconstruction. The research state of the art with this respect is promising but there is still room to improve. In particular, future strategies that combine both shape and appearance information would be of great interest. Non-linear representations of appearance might help improve the quality of appearance synthesis as a next step. Following [55] and [56], more scalable learning of animation synthesis [28] for 3D shapes might also be considered in the future if larger datasets are available, in order to ease the burden of surface capture and allow more automation of this type of animation.

7 CONCLUSION

We presented a solution for 4D surface capture animation using a novel organizing data structure, the extension of some methods of Mocap reuse to mesh data, and a global appearance representation of the animated character. The rendered results match the appearance quality of the source videos even after substantial dimension reduction. Although results are promising, popularizing this animation process is still a work in progress. However, applications such as 3D content generation for the media, virtual and augmented reality, or augmenting machine learning training sets provide great motivation for these solutions.

REFERENCES

[1] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan, "High-quality streamable free-viewpoint video," *ACM Trans. Graph.*, 2015.
[2] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor, P. Kohli, V. Tankovich, and S. Izadi, "Fusion4d: Real-time performance capture of challenging scenes," *ACM Trans. Graph.*, 2016.
[3] C. Stoll, J. Gall, E. De Aguiar, S. Thrun, and C. Theobalt, "Video-based reconstruction of animatable human characters," *ACM TOG*, vol. 29, no. 6, 2010.
[4] D. Vlasic, P. Peers, I. Baran, P. Debevec, J. Popović, S. Rusinkiewicz, and W. Matusik, "Dynamic shape capture using multi-view photometric stereo," *ACM TOG*, vol. 28, no. 5, 2009.
[5] J. Starck and A. Hilton, "Surface capture for performance-based animation," *IEEE CGA*, vol. 27, no. 3, 2007.

[6] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *CVPR*, 2015.

[7] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger, "Volumedeform: Real-time volumetric non-rigid reconstruction," in *ECCV*, 2016.

[8] R. Wang, L. Wei, E. Vouga, Q. Huang, D. Ceylan, G. Medioni, and H. Li, "Capturing dynamic textured surfaces of moving targets," in *ECCV*, 2016.

[9] J.-S. Franco and E. Boyer, "Efficient Polyhedral Modeling from Silhouettes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[10] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least squares conformal maps for automatic texture atlas generation," *ACM. Trans. Graph.*, 2002.

[11] B. Allain, J.-S. Franco, and E. Boyer, "An efficient volumetric framework for shape tracking," in *CVPR*, 2015.

[12] V. Tsiminaki, J.-S. Franco, and E. Boyer, "High resolution 3d shape texture from multiple videos," in *CVPR*, 2014.

[13] M. Botsch and O. Sorkine, "On linear variational surface deformation methods," *IEEE transactions on visualization and computer graphics*, 2008.

[14] A. Boukhayma and E. Boyer, "Video based Animation Synthesis with the Essential Graph," in *3DV*, 2015.

[15] A. Boukhayma, V. Tsiminaki, J.-S. Franco, and E. Boyer, "Eigen appearance maps of dynamic shapes," in *ECCV*, 2016.

[16] D. Casas, M. Volino, J. Collomosse, and A. Hilton, "4D Video Textures for Interactive Character Appearance," *Comput. Graph. Forum*, 2014.

[17] F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe, "Motion graphs for unstructured textured meshes," *ACM Trans. Graph.*, 2016.

[18] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Trans. Graph.*, 2002.

[19] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graph.*, 2002.

[20] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples." *ACM Trans. Graph.*, 2002.

[21] A. Safonova and J. K. Hodgins, "Construction and optimal search of interpolated motion graphs," *ACM Trans. Graph.*, 2007.

[22] L. Zhao and A. Safonova, "Achieving good connectivity in motion graphs," in *SCA*, July 2008.

[23] C. Ren, L. Zhao, and A. Safonova, "Human motion synthesis with optimization-based graphs." *Comput. Graph. Forum*, 2010.

[24] J. Starck, G. Miller, and A. Hilton, "Video-based character animation," in *SCA*, 2005.

[25] P. Huang, A. Hilton, and J. Starck, "Human motion synthesis from 3d video," in *CVPR*, 2009.

[26] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," *ACM Trans. Graph.*, 2004.

[27] P. Huang, M. Tejera, J. Collomosse, and A. Hilton, "Hybrid skeletal-surface motion graphs for character animation from 4d performance capture," *ACM. Trans. Graph.*, 2015.

[28] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Trans. Graph.*, 2016.

[29] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Trans. Graph.*, 2017.

[30] L. Kovar and M. Gleicher, "Flexible automatic motion blending with registration curves," in *SCA*, 2003.

[31] J. Wang and B. Bodenheimer, "Synthesis and evaluation of linear motion transitions," *ACM Trans. Graph.*, 2008.

[32] C. Zitnick, S. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," in *ACM SIGGRAPH*, 2004.

[33] M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent, "Floating textures," *Comput. Graph. Forum*, 2008.

[34] T. Tung, "Simultaneous super-resolution and 3D video using graph-cuts," in *CVPR*, 2008.

[35] B. Goldlücke, M. Aubry, K. Kolev, and D. Cremers, "A super-resolution framework for high-accuracy multiview reconstruction," *International Journal of Computer Vision*, 2014.

[36] M. Volino, D. Casas, J. Collomosse, and A. Hilton, "Optimal representation of multiple view video," in *BMVC*, 2014.

[37] D. Casas, C. Richardt, J. Collomosse, C. Theobalt, and A. Hilton, "4D Model Flow: Precomputed Appearance Alignment for Real-time 4D Video Interpolation," *Comput. Graph. Forum*, 2015.

[38] F. Xu, Y. Liu, C. Stoll, J. Tompkin, G. Bharaj, Q. Dai, H.-P. Seidel, J. Kautz, and C. Theobalt, "Video-based characters: Creating new human performances from a multi-view video database," in *ACM SIGGRAPH*, 2011.

[39] M. Turk and A. Pentland, "Eigenfaces for recognition," *J. Cognitive Neuroscience*, 1991.

[40] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3d faces," in *ACM SIGGRAPH*, 1999.

[41] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2001.

[42] K. Nishino, Y. Sato, and K. Ikeuchi, "Eigen-texture method: Appearance compression and synthesis based on a 3d model." *IEEE Trans. Pattern Anal. Mach. Intell.*, 2001.

[43] P. Huang, A. Hilton, and J. Starck, "Shape similarity for 3d video sequences of people," *Int. J. Comput. Vision*, 2010.

[44] D. Vlasic, I. Baran, W. Matusik, and J. Popović, "Articulated mesh animation from multi-view silhouettes," in *SIGGRAPH*, 2008.

[45] D. Q. Huynh, "Metrics for 3d rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, 2009.

[46] G. J. Székely, M. L. Rizzo, N. K. Bakirov *et al.*, "Measuring and testing dependence by correlation of distances," *The Annals of Statistics*, pp. 2769–2794, 2007.

[47] D. Xu, H. Zhang, Q. Wang, and H. Bao, "Poisson shape interpolation," in *ACM Symposium on Solid and Physical Modeling*, 2005.

[48] O. Sorkine and M. Alexa, "As-rigid-as-possible surface modeling," in *Eurographics Symposium on Geometry Processing*, 2007.

[49] M. Müller, *Information Retrieval for Music and Motion*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[50] J. Starck and A. Hilton, "Surface capture for performance-based animation," *IEEE Comput. Graph. Appl.*, 2007.

[51] C. Budd, P. Huang, M. Klaudiny, and A. Hilton, "Global non-rigid alignment of surface sequences," *Int. J. Comput. Vision*, 2013.

[52] D. Casas, M. Tejera, J. Guillemaut, and A. Hilton, "Interactive Animation of 4D Performance Capture," *Visualization and Computer Graphics, IEEE Transactions on*, 2013.

[53] J. Snchez Prez, E. Meinhardt-Llopis, and G. Facciolo, "TV-L1 Optical Flow Estimation," *Image Processing On Line*, 2013.

[54] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, 2004.

[55] A. Boukhayma, J.-S. Franco, and E. Boyer, "Surface Motion Capture Transfer with Gaussian Process Regression," in *CVPR*, 2017.

[56] A. Boukhayma and E. Boyer, "Controllable Variation Synthesis for Surface Motion Capture," in *3DV*, 2017.

Adnane Boukhayma received an engineering degree in digital communications from ENSEIRB and a master degree in signal and image processing from University of Bordeaux in 2013. He is now a Ph.D student at INRIA Grenoble specializing in computer vision and graphics.

Edmond Boyer is senior researcher at INRIA Grenoble (France) where he leads the Morpheo research team on the capture and the analysis of moving shapes using visual cues. His fields of competence cover computer vision, computational geometry and virtual reality. His current research interests are on 3D dynamic modelling from images and videos, motion perception and analysis from videos, and immersive and interactive environments. Edmond Boyer obtained a PhD in computer science from the Institut National Polytechnique de Lorraine in 1996. Edmond Boyer joined the INRIA Grenoble in 1998. From 1998 to 2010, he was associate professor of computer science at the university Joseph Fourier part of Grenoble universities. Edmond Boyer is co-founder of the 4DViews company that is specialized in 4D acquisition.